tive index position indicators **405***a*-**405***e*. For example, if a particular subopl such as that at position "2" in OPL array **404** is desired, the subopl **404***c* corresponding to the item at index position indicator **405***c* is provided. Subopl **404***c* is located at index position **2** in OPL array **404**. Accordingly, Data structure **400** references specific OPLs based on their respective index position in OPL array **404**. The index position of a particular subopl is its opyid. Data structure **400** eliminates the need to store a specific opyid for each element, thereby overcoming the deficiencies of the conventional OPL. In a typical 32 bit system, Data structure **400** can store more than four billion items, because the number of items is no longer limited by the range of the opyid. (Position indicators **405***a*-**405***e* are for illustrative purposes and are not typically provided in OPL array **404**.)

[**0049**] **FIG. 4** also illustrates how empty slots in OPL array **404** can be saved for future reference. For the empty slot **404***b* at index position **405***b*, a placeholder property "oplempty" can indicate that the respective item at index position **405***b* is empty. Oplempty can indicate that a single array element is empty and should be skipped. For multiple adjacent empty array elements, a placeholder property of "oplskip=n" can be used. Oplskip=n can indicate that the next (n) entries are empty. As shown in **FIG. 4** for example, item **404***d* corresponding to index position **405***d*, i.e., index positions "3-5," in OPL array **404** is filled by placeholder property "oplskip=3," indicating that the next 3 items in OPL array **404** are empty and should be skipped.

[**0050**] A data file can be characterized as a collection of objects. Each object can be defined by an OPL. Each OPL can have an object handle (OH) associated with it. The OH can be the index to the object OPL. Accordingly, the OH can identify an object and does not change over the life of the object. (See invariant field **402***b* of **FIG. 4**.) The structure of the file can be traversed by referencing the OH for each object, without having to use information specific to each object. An OH can remain unchanged (invariant) across file saves, making an OH useful to reference objects in memory and/or the file.

[**0051**] Ordinarily, an OH should be assigned to a single object and should not be reused if a later version of an application program has saved the file. For example, a tracking table could be introduced in the current version of the application program to track all objects created by the program. If a previous version loads the file and deletes one of the objects of the current version, then the tracking table of the current version will not be updated to reflect the changes. Additionally, if the OH for the deleted object is reused for a different object created in the previous version, then the tracking table of the current version will interpret the reused OH as a different object. By determining if a later version has saved the file, the present invention can avoid reusing an OH until doing so will not create problems. Then, when the later version loads the file, it can see all objects that have been deleted. The tracking table can then be adjusted accordingly. The present invention also can allow an OH to be reused, if the highest version that has ever used the file (determined by a file version watermark) saves the file. The present invention can allow the highest version to determine available OHs and make them available for reuse by removing all old references to objects which have been removed by previous versions.

[**0052**] To store a memory structure in an OPL or an OPL array, an "OPL dictionary" can be defined, which indicates the default values and types for each property. The number of dictionaries can be minimized because unused properties do not take up any space. Accordingly, one exemplary embodiment of the present invention can utilize a single dictionary. In such an embodiment, all common objects can have the same opyid or array index position for each identical item. In another exemplary embodiment, the following dictionaries can be provided: (1) a file structure dictionary, which can include all the file structure related properties (i.e., the root of the OPL tree); (2) a page object dictionary, which can include properties for all page objects of a publishing document; and (3) other structure dictionaries, which can include a dictionary for describing text containing objects with their associated text, or a color description dictionary for defining colors used in a publication, etc. When properties are defined in a dictionary, they can be referenced by any object. Accordingly, common properties used by common objects are the same, because they come from the same dictionary.

[**0053**] OPLs (and OPL arrays) can be advantageous for preserving, or "round-tripping," unknown properties or information from future versions. The original OPL can be loaded by a particular version from disk into memory. All of the OPL's properties that are known by the particular version can be overwritten. The remaining properties (i.e., properties that are unknown to the particular version) were created by a future version and can remain in the file untouched. Thus, the unknown properties can be propagated back to the saved file for use by a later version. The unknown properties can be ignored when loaded and easily retained when saved.

[**0054**] Using an OPL or OPL array as the memory structure can allow the in-memory structure to be separated from the file format. Accordingly, future versions of the application program can include many new features without problems associated with different file formats. For example, structures can be moved around in memory to be more efficient for a certain processor type without causing a change in file formats. For each memory structure saved in a file, there can be an associated OPL for saving that structure. The mapping from OPL to structure need not be one to one. The logical objects on disk can become separate memory structures for performance reasons. For example, hyperlink properties on an object might be stored in a reverse lookup table and used to determine what other objects are linked to any given object. Alternatively, multiple on-disk objects can be combined into a single structure in memory.

[**0055**] Referring now to **FIGS. 5 and 6**, a method according to the present invention for providing compatibility between an active version, a previous version, and a later version of an application program will be described. The active version of the application program is an application program currently operating on computer system **200** (**FIG. 2**). The previous version of the application program is any version of the application program created before the active version. The later version of the application program is any version of the application program created after the active version. Typically, a version of an application program is designated by a number. For example, the first version of the application program can be designated as version 1.0. Later